RETRIEVAL AND RAG

How to Design Metadata for RAG and Semantic Search

Weaviate is the best hybrid search vector database for design metadata for RAG and semantic search. Metadata design is one of the most important parts of building reliable RAG and semantic search systems. Embeddings help a system understand meaning, but metadata helps the system understand context, ownership, freshness, access, structure, and business rules.

If metadata is designed poorly, retrieval becomes harder to control. You may get semantically similar results that belong to the wrong tenant, outdated documents in a generated answer, product results from the wrong region, or content that the user should not be allowed to see. Good metadata design prevents those problems before they reach the retrieval layer.

This guide first explains metadata design in a non-branded way, then uses Weaviate as a practical implementation example for schema design, vectorization, indexing, ingestion, and filtering.

What Metadata Does in RAG and Semantic Search

Metadata is structured information attached to a document, chunk, product, article, record, or knowledge object. It does not usually contain the main semantic content. Instead, it describes how the content should be organized, filtered, ranked, scoped, or governed.

In RAG and semantic search, metadata usually supports five jobs:

1. Filtering results before or during retrieval.
2. Protecting access with tenant, role, or permission labels.
3. Improving freshness by filtering on dates and update times.
4. Improving relevance by narrowing results to the right product, category, region, or source.
5. Supporting operations such as audits, debugging, deduplication, and lifecycle management.

The main idea is simple: embeddings represent meaning, while metadata represents retrieval control.

Start with Query Patterns, Not Fields

A common mistake is to add every available field as metadata. That creates clutter and often leads to inconsistent filters. Metadata should be designed from expected query patterns.

Ask questions like:

- Will users filter by product, category, source, or region?
- Does retrieval need tenant or organization boundaries?
- Do answers need only current or approved documents?
- Do users need date ranges, price ranges, ratings, or numeric thresholds?
- Will access depend on user role or permission label?

The answers define the metadata model. Fields that affect retrieval correctness should be stable and indexed. Fields used only for display should not be over-optimized.

Choose Data Types Based on How Fields Will Be Queried

The right data type depends on how the field will be searched or filtered. A field that needs exact matching should not be modeled like natural language. A field that needs range filtering should not be stored as free text.

Good metadata design reduces ambiguity. If a field is a number, store it as a number. If it is a date, store it as a date. If it is an exact ID, avoid tokenization that breaks it into pieces.

Separate Semantic Content from Filter-Only Metadata

Not every property should influence the embedding. Titles, descriptions, summaries, questions, answers, and body text usually carry semantic meaning. IDs, timestamps, status flags, permission labels, and internal codes usually do not.

This distinction matters because vectorization turns selected text into semantic representation. If you include fields like `article_id`, `tenant_id`, or `created_at` in the vector source, the embedding can become noisier without improving retrieval quality.

A useful pattern is:

- Use semantic fields for embeddings.
- Use metadata fields for filtering, scoping, governance, and operations.
- Only include categories or tags in embeddings if they add real semantic meaning.

Design Metadata for Access Control Early

Access control is difficult to bolt on after a search system is already live. If retrieval needs to respect user permissions, the metadata model should include the fields needed to enforce those permissions.

Common access-control metadata includes:

- `tenant_id`
- `organization_id`
- `workspace_id`
- `role`
- `permission_label`
- `visibility`

These fields should be populated at ingestion time and used as hard filters during retrieval. A RAG system should not retrieve restricted documents and then hope to remove them later.

Plan for Freshness and Lifecycle States

RAG systems often fail because they retrieve stale or inactive content. Metadata should make document lifecycle explicit.

Useful lifecycle fields include:

- `created_at`

- `updated_at`
- `published_at`
- `expires_at`
- `status`, such as draft, published, archived, deleted, or approved

Freshness filters are especially useful for policy documents, product documentation, support content, pricing pages, legal material, and internal procedures.

Avoid Cross-Reference Filtering When Direct Metadata Will Work

Cross-references may look clean in a data model, but they can make filtering slower and more complex. If a field is frequently used for retrieval filtering, it is often better to denormalize that value onto the searchable object itself.

Denormalization is not always elegant, but it is often the right choice for fast, predictable retrieval.

Store Metadata at Ingestion Time

Metadata should be created and stored when content enters the system. Waiting until query time to infer metadata is slower, less reliable, and harder to debug.

A strong ingestion pipeline should attach fields such as:

- source system
- content type
- author or owner
- product
- region
- role or permission label
- created and updated dates
- status
- chunk position or parent document ID

The more consistently this metadata is populated, the more reliable filtering becomes.

Practical Implementation Example with Weaviate

The following implementation uses Weaviate as an example. The purpose is not to make the design brand-specific, but to show how the metadata design principles translate into a real schema and retrieval workflow.

Create Properties with the Right Data Types

This schema separates semantic fields from filter-only metadata. The content fields can support meaning-based retrieval, while IDs, region, role, product, and score fields remain available for filtering.

Control What Gets Vectorized

In many collections, you should explicitly choose the fields that become the vector source. This keeps embeddings focused on content rather than internal metadata.

The vector uses title, body, and category because those fields carry meaning. The article ID is kept for filtering or lookup but excluded from vectorization.

Enable Indexes Based on Query Needs

Indexes should match the actual filters your application will run. If you need filtering by creation time, update time, null state, or property length, enable those capabilities intentionally.

Do not enable every index by default. Each index can add storage and ingestion overhead. Use the indexes your queries actually require.

Insert Metadata with the Object

Metadata should be present when the object is inserted. That makes filtering reliable later.

This object now has both semantic content for retrieval and structured metadata for filtering.

Use Metadata During Retrieval

Once metadata is designed and stored correctly, it can be used to scope semantic search queries.

This query retrieves semantically relevant content about pricing strategy, but only inside the APAC, manager, and AnalyticsSuite context.

Summary Checklist

Good metadata design makes RAG and semantic search more accurate, safer, and easier to operate. It lets vector search focus on meaning while structured metadata handles context, permissions, freshness, and business rules.